REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUGUST 2011	Conference Paper-Post Print	June 2009 – March 2011

4. TITLE AND SUBTITLE

SECURE PROACTIVE RECOVERY-A HARDWARE BASED MISSION ASSURANCE SCHEME

5a. CONTRACT NUMBER
In House

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Ruchika Mehresh, Shambhu Upadhyaya, and Kevin Kwiat

5d. PROJECT NUMBER
23G4

5e. TASK NUMBER
IH

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFRL/Information Directorate
Rome Research Site/RIGD
525 Brooks Road
Rome, NY 13441

University of Buffalo
Dept of Computer Science and Engineering
Buffalo, NY 14260

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/Information Directorate
Rome Research Site
26 Electronic Parkway
Rome NY 13441

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2011-2

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA #: 88ABW-2010-6904
Date Cleared: November 16, 2010.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Mission Assurance in critical systems entails both fault tolerance and security. Since fault tolerance via redundancy or replication is contradictory to the notion of a limited trusted computing base, normal security techniques cannot be applied to fault tolerant systems. Thus, in order to enhance the dependability of mission critical systems, designers employ a multi-phase approach that includes fault/threat avoidance/prevention, detection and recovery. Detection phase is the fallback plan for avoidance/prevention phase, as recovery phase is the fallback plan for detection phase. However, despite this three-stage barrier, a determined adversary can still defeat system security by staging an attack on the recovery phase. Recovery being the final stage of the dependability life-cycle, unless certain security methodologies are used, full assurance to mission critical operations cannot be guaranteed. For this reason, a new methodology is proposed: secure proactive recovery.

15. SUBJECT TERMS

Security, fault tolerance, mission assurance, critical systems, hardware

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kevin A. Kwiat
U	U	U	UU	10	19b. TELEPHONE NUMBER (*Include area code*) N/A

Secure Proactive Recovery – a Hardware Based Mission Assurance Scheme

Ruchika Mehresh[1], Shambhu Upadhyaya[1] and Kevin Kwiat[2]
[1]State University of New York at Buffalo, USA
[2]Air Force Research Laboratory, Rome, USA
rmehresh@buffalo.edu
shambhu@buffalo.edu
kwiatk@rl.af.mil

Abstract: Mission Assurance in critical systems entails both fault tolerance and security. Since fault tolerance via redundancy or replication is contradictory to the notion of a limited trusted computing base, normal security techniques cannot be applied to fault tolerant systems. Thus, in order to enhance the dependability of mission critical systems, designers employ a multi-phase approach that includes fault/threat avoidance/prevention, detection and recovery. Detection phase is the fallback plan for avoidance/prevention phase, as recovery phase is the fallback plan for detection phase. However, despite this three-stage barrier, a determined adversary can still defeat system security by staging an attack on the recovery phase. Recovery being the final stage of the dependability life-cycle, unless certain security methodologies are used, full assurance to mission critical operations cannot be guaranteed. For this reason, we propose a new methodology, viz. secure proactive recovery that can be built into future mission-critical systems in order to secure the recovery phase at low cost. The solution proposed is realized through a hardware-supported design of a consensus protocol. One of the major strengths of this scheme is that it not only detects abnormal behavior due to system faults or attacks, but also secures the system in case where a smart attacker attempts to camouflage by playing along with the predefined protocols. This sort of adversary may compromise certain system nodes at some earlier stage but remain dormant until the critical phase of the mission is reached. We call such an adversary The Quiet Invader. In an effort to minimize overhead, enhance performance and tamper-proof our scheme, we employ redundant hardware typically found in today's self-testing processor ICs, like design for testability (DFT) and built-in self-test (BIST) logic. The cost and performance analysis presented in this paper validates the feasibility and efficiency of our solution.

Keywords: security, fault tolerance, mission assurance, critical systems, hardware

1. Introduction

Research in the past several decades has seen significant maturity in the field of fault tolerance. But, fault tolerant systems still require multi-phased security due to the lack of a strong trusted computing base. The first phase in this regard is avoidance/prevention, which consists of proactive measures to reduce the probability of any faults or attacks. This can be achieved via advanced design methodologies like encryption. The second phase, detection, primarily consisting of an intrusion detection system attempts to detect the faults and malicious attacks that occur despite the preventive measures. The final phase is the recovery that focuses on recuperating the system after the occurrence of attack/fault. Generally, fault tolerant systems rely on replication and redundancy for fault-masking and system recovery.

These three layers of security provide a strong defense for mission critical systems. Yet, if a determined adversary stages an attack on the recovery phase of an application, it is quite possible that the mission will fail due to the lack of any further countermeasures. Therefore, these systems need the provisioning of another layer of defense to address attacks that may be brought about by malicious opponents during the recovery phase itself.

The quiet invader is another serious threat that we consider. Attacking the mission in its critical phase not only leaves the defender with less time to respond, but cancelling the mission at this late stage is far more expensive than cancelling it at some earlier stage. In the case where the defender is not left with enough time to respond to the attack, it can lead to major economic loss and even fatalities.

We develop a framework for mission assured recovery using the concept of runtime node-to-node verification implementable at low-level hardware that is not accessible by the adversary. The rationale behind this approach is that if an adversary can compromise a node by gaining root privilege to user-space components, any solution developed in the user space will not be effective since such solutions may not remain secure and tamper-resistant. In our scheme, the entire verification process can be carried out in a manner that is oblivious to the adversary, which gains the system an additional

advantage. We explore the potential of utilizing the test logic on the processors (and hence the name "hardware-based mission assurance scheme") for implementing our secure proactive recovery paradigm. This choice makes our solution extremely cost effective. In order to establish the proof-of-concept for this proposal, we will consider a simple mission critical system architecture that uses majority consensus for diagnosis and recovery. Finally, we analyze the security, usability and performance overhead for this scheme.

2. Related work

The solutions proposed in the literature to address faults/attacks in fault tolerant systems are designed to employ redundancy, replication and consensus protocols. They are able to tolerate the failure of up to *f* replicas. However, given enough time and resources, an attacker can compromise more than *f* replicas and subvert the system. A combination of reactive and proactive recovery approaches can be used to keep the number of compromised replicas under *f* at all times (Sousa et al. 2007). However, as the attacks become more complex, it becomes harder to detect any faulty or malicious behavior (Wagner and Soto 2002). Moreover, if one replica is compromised, the adversary holds the key to other replicas too. To counter this problem, researchers have proposed spatial diversity in software. Spatial diversity can slow down an adversary but eventually the compromise of all diverse replicas is possible. Therefore, it was further proposed to introduce time diversity along with the spatial diversity. Time diversity modifies the state of the recovered system (OS access passwords, open ports, authentication methods, etc.). This is to assure that an attacker is unable to exploit the same vulnerabilities that he had exploited before (Bessani et al. 2008).

3. Threat model

We developed an extensive threat model to analyze security logically in a wide range of scenarios. Assume that we have *n* replicas in a mission-critical application and the system can tolerate the failure of up to *f* replicas during the entire mission.

Scenario 1: Attacks on Byzantine fault-tolerant protocols

Assume that no design diversity is introduced in a replicated system. During the mission lifetime, an adversary can easily compromise *f+1* identical replicas and bring down the system.

Scenario 2: Attacks on proactive recovery protocols

In proactive recovery, the whole system is rejuvenated periodically. However, the adversary becomes more and more knowledgeable as his attacks evolve with each succeeded/failed attempt. So it is only a matter of time before he is able to compromise *f+1* replicas between periodic rejuvenations. Furthermore, the compromised replicas can disrupt the system's normal functioning in many ways like creating extra traffic so the recovery is delayed and the adversary gains more time to compromise *f+1* replicas (Sousa et al. 2007).This is a classic case of attacking the recovery phase.

Scenario 3: Attacks on proactive-reactive recovery protocols

Proactive-reactive recovery solves several major problems, except that if the compromised node is recovered by restoring the same state that was previously attacked, the attacker will already know the vulnerabilities (Sousa et al. 2007). In this case, a persistent attacker may get faster with time, or may invoke many reactive recoveries exhausting the system resources. Large number of recoveries also affects the system availability adversely. This is also an instance of attacking the recovery phase. Furthermore, arbitrary faults are very difficult to detect (Haeberlen et al. 2006).

Scenario 4: Attacks on proactive-reactive recovery with spatial diversity

Spatial diversity in replicas is proposed to be a relatively stronger security solution. It can be difficult and more time-consuming for the adversary to compromise *f+1* diverse replicas but it is possible to compromise these diverse replicas eventually, especially for long running applications. Also, most of the existing systems are not spatially diverse. Introducing spatial diversity into the existing systems is expensive.

Time diversity has been suggested to complement the spatial diversity so as to make it almost impossible to predict the new state of the system (Bessani et al. 2008). The complexity involved in

implementing time diversity in a workable solution is very high because it will have to deal with on-the-fly compatibility issues and much more. Besides, updating replicas and other communication protocols consume considerable time and resources. A decent workable solution employing space diversity still needs a lot of work (Banatre et al. 2007), so employing time diversity is a step planned too far into the future.

Scenario 5: The quiet invader

In the presence/absence of spatial diversity, an adversary may be able to investigate a few selected nodes quietly and play along with the protocol to avoid getting caught and gain more time to understand the system. After gathering enough information, the adversary can design attacks for *f+1* replicas and launch the attacks on all of them at once when he is ready or when the mission enters a critical stage. If these attacks are not detected or dealt with in time, the system fails. This is an evasive attack strategy for subverting the detection and recovery phases. Similar threat models have been discussed in literature previously (Todd et al. 2007, Del Carlo 2003).

Scenario 6: The physical access threat

Sometimes system nodes are deployed in an environment where physical access to them is a highly probable threat. For instance, in the case of wireless sensor network deployment, sensor nodes are highly susceptible to physical capture. To prevent such attacks, we need to capture any changes in the physical environment of a node. A reasonable solution may involve attaching motion sensors to each node. Any unexpected readings from these motion sensors will indicate a possible threat and then our scheme can be used to assure the mission.

4. System design

4.1 Assumptions

We work with a simplified, centralized architecture of a mission critical application in order to describe and evaluate the proposed scheme. No spatial or time diversity is assumed, though our scheme will work with any kind of diversity.

The network can lose, duplicate or reorder messages but is immune to partitioning. The coordinator (central authority and trusted computing base) is responsible for periodic checkpointing in order to maintain a consistent global state. The stable storage at coordinator holds the recovery data through all the tolerated failures and their corresponding recoveries. We assume sequential and equidistant checkpointing (Elnozahy et al. 2002).

The replicas are assumed to be running on identical hardware platforms. Each node has advanced CPU (Central processing unit) and memory subsystems along with the test logic (in the form of DFT and BIST) that is generally used for manufacture test. Refer to Fig. 1(a). All the chips comply with the IEEE 1149.1 JTAG standard (Abramovici and Stroud 2001). Fig. 1(b) elaborates the test logic and boundary scan cells corresponding to the assumed hardware.

We assume a software tripwire running on each replica that can be used to detect a variety of anomalies at the host. By instrumenting the openly available tripwire source code (Hrivnak 2002), we can direct the "intrusion alert/alarm" to a set of system registers (using low level coding).The triggered and latched hardware signature will be read out by taking a snapshot of the system registers using the “scan-out” mode of the observation logic associated with the DFT hardware. The bit pattern will be brought out to the CPU ports using the IEEE 1149.1 JTAG instruction set in a tamper-resistant manner. Once it is brought out of the chip, it will be securely sent to the coordinator for verification and further action. This way, the system will be able to surreptitiously diagnose the adversary’s action.

4.2 Conceptual basics

We present a simple and practical alternative to the spatial/time diversity solutions in order to increase the resilience of a fault tolerant system against benign faults and malicious attacks. In particular, this is to address the threat of a quiet invader (Scenario 5 of Section 3). An adversary needs to compromise *f+1* replicas out of the *n* correctly working replicas in order to affect the result of a majority consensus protocol and disrupt the mission.

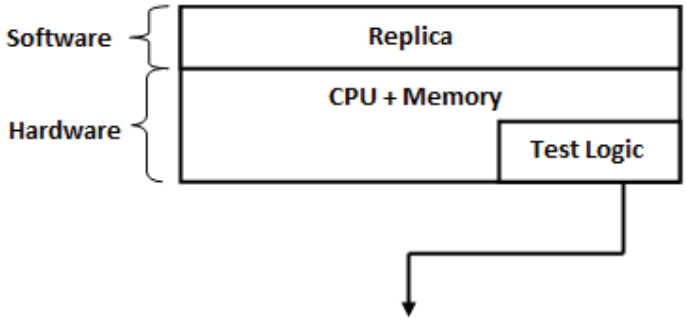

Figure 1(a): Replicated hardware

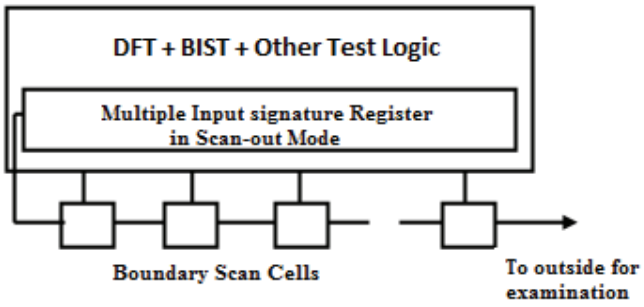

Figure 1(b): Capturing signature

The key idea is to detect a system compromise by a smart adversary who has taken over some replicas (or has gained sufficient information about them) but is playing along in order to gain more time. From the defender's point of view, if the system knows which of the *n* replicas have become untrustworthy, the mission can still succeed with the help of the surviving healthy replicas. Smart attackers try to minimize the risk of getting caught by compromising only the minimum number of replicas required in order to subvert the entire system. Aggressive attackers can be clearly and easily detected and thus their attacks can be recovered from. So a smart defender should be able to detect the attacks surreptitiously so as not to make the attacker aggressive. This especially holds for the cases when a smart attacker has been hiding for long and the mission is nearing completion. At this stage, the priority is not to identify the attacker but to complete the mission securely.

The proposed scheme offers a passive detection and recovery, in order to assure the adversary of its apparent success to prevent him from getting more aggressive. At some later stage, when the adversary launches an attack to fail *f+1* replicas at once, the attack fails because those replicas have already been identified and ousted from the voting process without the knowledge of the attacker. In our solution, we require that there should be at least two correctly working replicas to provide a duplex system at a minimum, for the mission to succeed. The advantage of this approach is that in the worst case where all the replicas are compromised, the system will not deliver a result, rather than delivering a wrong one. This is a necessary condition for many safety-critical missions. If an adversary can compromise a replica by gaining root privilege to user-space components, one should note that any solution developed in the user space will not be effective since such solutions will not remain secure and tamper-resistant. Therefore, our paradigm achieves detection of node compromise through a verification scheme implementable in low-level hardware. We use software or hardware-driven tripwires that would help detect any ongoing suspicious activity and trigger a hardware signature that indicates the integrity status of a replica. This signature is generated without affecting the application layer, and hence the attacker remains oblivious of this activity. Also, a smart attacker is not likely to monitor the system thoroughly as that may lead to detection. This signature is then securely collected and sent to the coordinator that performs the necessary action.

4.3 Checkpointing

In our simplified application, the checkpointing module that affiliates to the coordinator establishes a consistent global checkpoint and also carries out voting procedures that lead to anomaly detection due to faults, attacks or both.

The coordinator starts the checkpointing/voting process by broadcasting a request message to all the replicas, asking them to take checkpoints. It also initiates a local timer that runs out if the coordinator does not receive the expected number of replies within a specific time frame. On receiving this message, all the replicas pause their respective executions and take a checkpoint. These checkpoints are then sent over the network to the coordinator through a secure channel using encryption. On receiving the expected number of checkpoints, coordinator compares them for consistency. If all checkpoints are consistent, it broadcasts a commit message that completes the two-phase checkpoint protocol. After receiving the commit message, all the replicas resume their respective executions. This is how the replicas execute in lockstep. In case the timer runs out before the expected number of checkpoints are received at the coordinator, it sends out another request message. All the replicas send their last locally stored checkpoints as a reply to this request message. In our application, we have limited the number of repeated checkpoint requests to three per non-replying replica. If a replica does not reply to three (or a threshold count) checkpoint request messages, it is considered dead by the coordinator and a commit message is sent to the rest of the replicas if their checkpoints are consistent. In case that the checkpoints are not consistent, the coordinator replies with a rollback message to all the replicas. This rollback message includes the last consistent checkpoint that was stored on the stable storage at the coordinator. All the replicas then return to the previous state of execution as defined by the rollback message. If a certain replica fails to deliver consistent checkpoint and causes more than three (or a threshold count) consecutive rollbacks, the fault is considered permanent and the replica is excluded from the system.

A hardware signature is generated at each replica and piggybacked on the checkpoint when it is sent to the coordinator. This signature quantifies the integrity status of the replica since the last successful checkpoint. For simplicity, we use the values – all-0s (for an uncompromised replica) and all-1s (for a compromised replica). A host-based intrusion detection sensor at all the replicas is responsible for generating these signatures. If the coordinator finds any hardware signature to be all-1s, then the corresponding replica is blacklisted and any of its future results/checkpoints are ignored at the coordinator. However, the coordinator continues normal communication with the blacklisted replica to keep the attacker unaware of this discovery.

Finally, all the results from each of the non-blacklisted replicas will be voted upon by the coordinator for the final result.

4.4 Using built-in test logic for hardware signature generation and propagation

As described under assumptions, the system uses a software-driven trip-wire that monitors the system continuously for a specified range of anomalies. Tripwire raises an alarm on anomaly detection by setting the value of a designated system register to all-1s (it will be all-0s otherwise). This value then becomes the integrity status indicator for the replica and is read out using the scan-out mode of the test logic. It is then securely sent to the coordinator for verification.

5. Performance analysis

Most of the mission critical military applications that employ checkpointing or proactive security tend to be long running ones. For instance, a rocket launch countdown running for hours/days. Therefore, our performance analysis will focus on long running applications and their overall execution time.

Since our scheme employs built-in hardware for implementing security, and security-related notifications piggyback the checkpointing messages, our security comes nearly free for systems that already use checkpointing for fault tolerance. However, many legacy systems that do not use any checkpointing will need to employ checkpointing before they can benefit from our scheme. In such cases, cost of checkpointing is also included in the cost of employing our security scheme. To cover all these possibilities, we consider the following three cases.

Case 1: This case includes all the mission critical legacy systems that do not employ checkpointing or security.

Case 2: This case examines mission critical systems that employ checkpointing as a safety measure in the absence of any failures or attacks. Note that this will be the worst case scenario for Case 1 systems that may adopt our scheme because there are practically no faults/attacks. Also, our security scheme is nearly free for Case 2 systems, if they choose to employ it.

Case 3: The systems considered under Case 3 employ checkpointing and our proposed security scheme (hardware signature verification). This case considers the occurrence of failures and security-related attacks.

These three cases allow us to study the cost of adopting our security scheme in all possible scenarios.

Since the proposed system is composed of both hardware and software subsystems, we could not use one standard simulation engine to simulate the entire application accurately and obtain data. Therefore, we combined the results obtained from individually simulating the software and the hardware components using our multi-step simulation approach (Mehresh et al. 2010).

5.1 Simplified system prototype development

Figure 2 shows the modular design of the simplified system for mission critical applications with *n* replicas. The coordinator is the core of this centralized replicated system. It is responsible for voting operations on intermediate results, integrity signatures and checkpoints obtained from the replicas. The heartbeat manager broadcasts periodic ping messages to determine if the nodes are alive. The replicas are identical copies of the workload executing in parallel in lockstep.

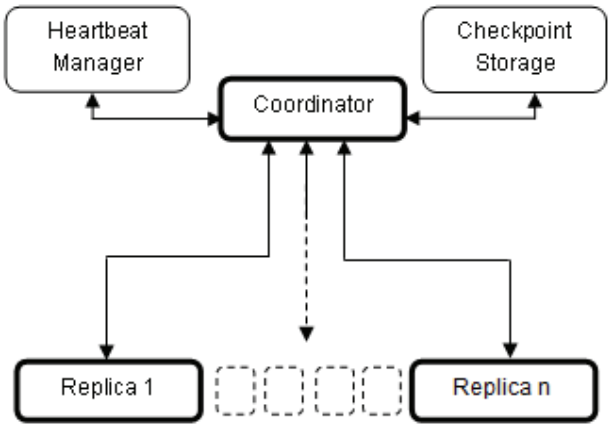

Figure 2: Overall system design

5.2 Multi-step simulation approach

We use a multi-step simulation approach to evaluate the system performance for the three cases. This new approach is required because there are currently no benchmarks for evaluating such systems. A combination of pilot system implementation and simulation is used to obtain more realistic and statistically accurate results.

Different components of this evaluation include a JAVA implementation based on Chameleon ARMORs (Kalbarczyk et al. 1999), ARENA simulation (http://www.arenasimulation.com/) and CADENCE simulation (http://www.cadence.com). ARENA simulation is discrete event and it simulates the given system at a high level of abstraction. The lower levels of abstraction that become too complex to model are parameterized using the data obtained from conducting experiments with the JAVA system prototype. Another reason for using ARENA simulator is the analysis of long running mission critical applications. Such an analysis with real-time experiments is not efficient and extremely time consuming. The Java prototype consists of socket programming across a network of 100 Mbps bandwidth. The experiments for measuring performance were conducted on Windows platform with an Intel Core Duo 2 GHz processor and 2 GB RAM. CADENCE simulation is primarily used for the feasibility study of the proposed hardware scheme. To verify the precision of our simulators, test cases were developed and deployed for the known cases of operation.

This system accepts workloads from the user and executes them in a fault tolerant environment. We used the Java SciMark 2.0 workloads as user inputs in this system prototype. The four workloads that we used are: Fast Fourier Transform (FFT), Jacobi Successive Over-relaxation (SOR), Sparse Matrix

multiplication (Sparse) and Dense LU matrix Factorization (LU). The standard large data sets (http://math.nist.gov/scimark2) were used.

Data-sets from short running replicated experiments were collected and fitted probability distributions were obtained using ARENA input data analyzer. These distributions defined the stochastic parameters for ARENA simulation model.

We examine the feasibility of the hardware component of this architecture (as described under assumptions) as follows. The integrity signature of a replica is stored in the flip flops of the boundary scan chain around a processor. This part of our simulation is centered on a boundary scan inserted DLX processor (Patterson and Hennessy 1994). Verilog code for the boundary scan inserted DLX processor is elaborated in cadence RTL compiler. To load the signature into these scan cells a multiplexer is inserted before each cell, which has one of the inputs as test data input (TDI) and the other from the 32 bit signature vector. Depending on the select line either the test data or the signature is latched into the flip flops of the scan cells. To read the signature out the bits are serially shifted from the flip flops onto the output bus.

5.3 Results

We analyze the prototype system for the three cases described earlier. Since we want to evaluate the performance of this system in the worst case scenario where the checkpointing overhead is maximum, we choose sequential checkpointing (Elnozahy et al. 2002). For the following analysis (unless mentioned), checkpoint interval is assumed to be 1 hour. Table 1 presents the execution times for the four Scimark workloads. The values from Table 1 are plotted in Figure 3 on a logarithmic scale. We can see that the execution time overhead increases a little when the system shifts from Case 1 to Case 2 (i.e., employing our scheme as a preventive measure). However, the execution time overhead increases rapidly when the system moves from Case 2 and Case 3. The execution overhead will only increase substantially if there are too many faults present, in which case it would be worth the fault tolerance and security that comes along. As we can see from the values of Table 1, an application that runs for 13.6562 hours will incur an execution time overhead of only 13.49 minutes in moving from Case 1 to Case 2.

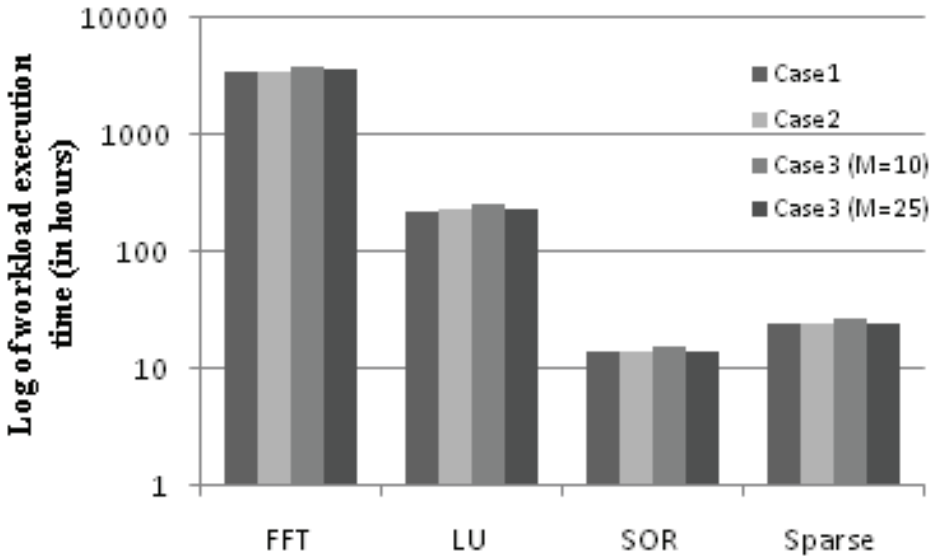

Figure 3: Execution times for Scimark workloads across three cases, on a logarithmic scale

Figure 4 shows the percentage increase in execution times of various workloads when the system upgrades from a lower case to a higher one. It is assumed that these workload executions do not have any interactions (inputs/outputs) with the external environment. The percentage increase in execution times of all the workloads when the system upgrades from Case 1 to Case 2 is only around 1.6%. An upgrade from Case 1 to Case 3 (with mean time to fault, M =10) is around 9%. These percentages indicate acceptable overheads.

Table 1: Execution times (in hours) for the Scimark workloads across three cases

	FFT	LU	SOR	Sparse
Case 1	3421.09	222.69	13.6562	23.9479
Case 2	3477.46	226.36	13.8811	24.3426
Case 3 (M=10)	3824.63	249.08	15.2026	26.7313
Case 3 (M=25)	3593.39	233.83	13.8811	24.3426

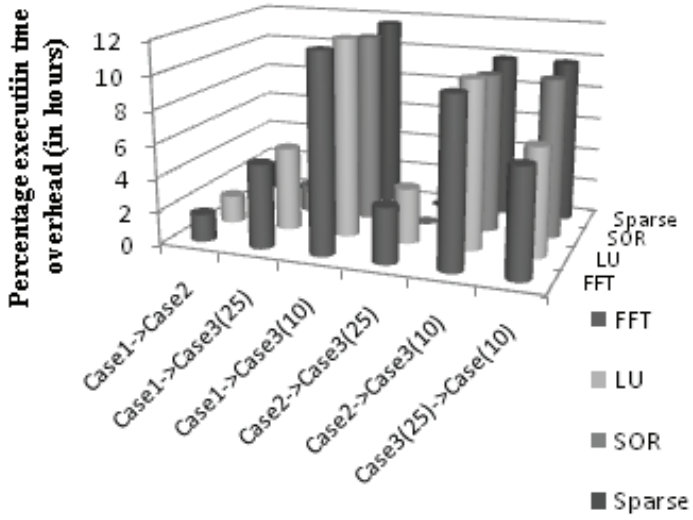

Figure 4: Percentage execution time overheads incurred by the Scimark workloads while shifting between cases

As Table 1 shows, for a checkpoint interval of 1 hour and M =10, the workload LU executes for approximately 10 days. Figure 5 shows the effect of increasing checkpoint interval for workload LU for different values of M ranging from 5 to 25. The optimal checkpoint interval values (and the corresponding execution times) for the graph plots in Figure 5 are provided in Table 2.

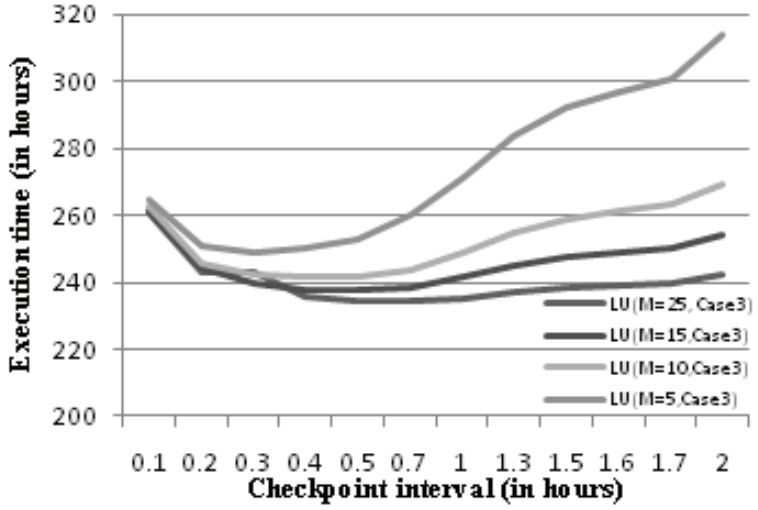

Figure 5: Effect of checkpoint interval on workload execution times at different values of M

Note that we used the multi-step approach for this simulation and the parameters for the simulation model were derived from experimentation. Therefore, these results do not just represent the data trends but are also close to the statistically expected real-world values.

Table 2: Approximate optimal checkpoint interval values and their corresponding workload execution times for LU (Case 3) at different values of M

	M=5	M=10	M=15	M=25
Optimal Checkpoint Interval (hours)	0.3	0.5	0.65	0.95
Execution Times(hours)	248.97	241.57	238.16	235.06

6. Conclusion

This paper proposes a hardware based proactive solution to secure the recovery phase of mission critical applications. A detailed threat model is developed to analyze the security provided by our scheme. The biggest strengths of this research is its ability to deal with smart adversaries, give priority to mission assurance, and use redundant hardware for capturing integrity status of a replica outside the user space. Since this scheme is simple and has no visible application specific dependencies, its implementation has the potential to be application transparent. For performance evaluation, we investigated a simplified mission critical application prototype using a multi-step simulation approach. We plan to enhance the centralized architecture to a distributed system for our future research work. We defined cases to investigate the cost involved in applying our security scheme to all kinds of systems (including the legacy systems with no fault tolerance). The performance evaluation showed promising results and the cost/performance overhead is only a small percentage of the original execution times when faults are absent. As the rate of fault occurrence increases, the overhead increases too, but this additional overhead comes with fault tolerance and security. Overall, we believe that our solution provides strong security at low cost for mission critical applications.

Acknowledgments

This work was supported in part by ITT Grant No. 200821J. This paper has been approved for Public Release; Distribution Unlimited: 88ABW-2010-6094 dated 16 Nov 2010.

References

Abramovici, M. and Stroud, C.E. (2001) "BIST-based test and diagnosis of FPGA logic blocks", IEEE Transactions on VLSI Systems, volume 9, number 1, pages 159-172, February.

Banatre, M., Pataricza, A., Moorsel, A., Palanque, P. and Strigini, L. (2007) From resilience-building to resilience-scaling technologies: Directions – ReSIST, NoE Deliverable D13. DI/FCUL TR 07–28, Dep. Of Informatics, Univ. of Lisbon, November.

Bessani, A., Reiser, H.P., Sousa, P., Gashi, I., Stankovic, V., Distler, T., Kapitza, R., Daidone, A. and Obelheiro, R. (2008) “FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery”, Proceedings of the ACM Middleware'08 companion, December.

Del Carlo, C. (2003) Intrusion detection evasion, SANS Institute InfoSec Reading Room, May.

Elnozahy, E.N., Alvisi, L., Wang, Y. and Johnson, D.B. (2002) "A survey of rollback-recovery protocols in message-passing systems", ACM Computing Surveys (CSUR), volume 34 number 3, pages 375-408, September.

Haeberlen, A., Kouznetsov, P. and Druschel, P. (2006) “The case for Byzantine fault detection”, Proceedings of the 2nd conference on Hot Topics in System Dependability, volume 2, November.

Hrivnak, A. (2002) *Host Based Intrusion Detection: An Overview of Tripwire and Intruder Alert,* SANS Institute InfoSec Reading Room, January.

Kalbarczyk, Z., Iyer, R.K., Bagchi, S. and Whisnant, K. (1999) "Chameleon: a software infrastructure for adaptive fault tolerance", IEEE Transactions on Parallel and Distributed Systems, volume 10, number 6, pages 560-579, June.

Mehresh, R., Upadhyaya, S. and Kwiat, K. (2010) “A Multi-Step Simulation Approach Toward Fault Tolerant system Evaluation”, Third International Workshop on Dependable Network Computing and Mobile Systems, October.

Patterson, D. and Hennessy, J. (1994) Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann.

Sousa, P., Bessani, A., Correia,M., Neves, N.F. and Verissimo, P. (2007) “Resilient intrusion tolerance through proactive and reactive recovery”, Proceedings of the 13th IEEE Pacific Rim Int. Symp. on Dependable Computing, pages 373–380, December.

Todd, A.D., Raines, R.A., Baldwin, R.O., Mullins, B.E. and Rogers, S.K. (2007) “Alert Verification Evasion Through Server Response Forging”, Proceedings of the 10th International Symposium, RAID, pages 256-275, September.

Wagner, D. and Soto, P. (2002) “Mimicry attacks on host-based intrusion detection systems”, *Proceedings of the 9th ACM conference on Computer and communications security*, November.